Description

REAL-TIME TRAINING FOR A COMPUTER CODE INTRUSION DETECTION SYSTEM

Inventors: Carey Nachenberg and Frank Barajas

Technical Field

This invention pertains to the field of thwarting intrusions perpetrated by malicious attackers to computer code (e.g., databases).

Background Art

The background art includes systems where the computer code being attacked is a database. Such systems are called database intrusion detection systems. Some of these systems utilize offline non-real-time training in order to detect suspicious or anomalous activity. These systems use database log files external to the database, or audit tables internal to the database, in conjunction with the training. Since these database intrusion detection systems are trained during normal usage of the database, the amount of logged data may be extensive -- potentially many gigabytes in size over the course of the training. Requiring the system administrator to collect multiple gigabytes of logged data to train the system is expensive in terms of storage needs, and requires constant monitoring by the system administrator to ensure that the logged entries do not fill up the storage area of the computer used for the training. In addition, many database systems override older logged entries

when the number of entries exceeds a pre-allocated storage limit. This causes entries to be overwritten and thus lost before training can occur. Furthermore, if the audit logs are stored within the database itself (to be used later during an operational step), such storage uses up valuable resources on the database and negatively impacts the database's performance.

Examples of offline non-real-time database intrusion detection systems are described in *Lee, et al.*, "Learning Fingerprints for a Database Intrusion Detection System", ESORICS 2002, pp. 264-279, published in November 2002 by Springer-Verlag, Berlin and Heidelberg, Germany; and *C. Chung*, et al., "DEMIDS: A Misuse Detection System for Database Systems", Department of Computer Science, University of California at Davis, Davis, California, October 1, 1999.

Disclosure of Invention

The present invention comprises systems, methods, and computer-readable media for training a computer code intrusion detection system in real time. A method embodiment of the present invention comprises the steps of observing (22), in real time, commands (5) that are accessing the computer code (1); and deriving (23) from said commands (5), in real time, a set (6) of acceptable commands.

Brief Description of the Drawings

These and other more detailed and specific objects and features of the present invention are more fully disclosed in the following specification, reference being had to the accompanying drawings, in which:

Figure 1 is a block diagram illustrating embodiments of the present invention.

Figure 2 is a flow diagram illustrating method embodiments of the present invention.

Detailed Description of the Preferred Embodiments

This invention has applicability to any code intrusion detection system, i.e., any system in which computer code 1 is susceptible to being attacked by commands 5 which may be malicious, due to malicious intent on the part of the user 3 who issues the command 5. As used herein, "user" can refer to a client computer 3 and/or to a human who has control of computer 3. As illustrated in Figure 1, there can be a plurality N of users 3, where N is any positive integer.

Most of the following description illustrates the special case where the computer code 1 is a database 1. Database 1 can be any type of database, such as a relational database or a flat file. When database 1 is a relational database, commands 5 are typically written in a SQL language. As used herein, "SQL" is taken in the broad sense to mean the original language known as

SQL (Structured Query Language), any derivative thereof, or any structured query language used for accessing a relational database. In the case where computer code 1 is not a relational database, the commands can be written in another language such as XML. Database 1 may have associated therewith an internal audit table 11 and/or an external database log file 12 for storing audit and/or ancillary information pertaining to database 1. Database 1 is typically packaged within a dedicated computer known as a database server 2, which may also contain database communications module 15 and other modules not illustrated. Modules 1, 11, 12, 15, 13, 17, 4, 7, 6, 8, and 9 can be implemented in software, firmware, hardware, or any combination thereof, and are typically implemented in software.

Computer code intrusion detection system 19 (and its special case, database intrusion detection system 19) encompasses modules 17, 4, 13, 7, 6, 8, and 9. Figure 1 illustrates the case where these modules are stand-alone modules separate from database server 2. However, they could just as well be incorporated within database server 2, e.g., they could be incorporated within database communications module 15. Thus, intrusion detection system 19 could be published by a third party as a standalone package on any type of computer-readable medium, or bundled by the manufacturer of the database 1 with module 15. The purpose of intrusion detection system 19 is to protect computer code 1

from users 3 that have nefarious intent. For example, such users may desire to steal credit card information from database 1, modify database 1 to credit their account with additional money, modify database 1 to spread a computer worm, etc.

With respect to Figure 2, it can be seen that intrusion detection system 19 operates in three phases. Phase one is a training phase and encompasses steps 21 through 24. Phase two is an optional reporting phase and encompasses step 25. Phase 3 is an operational phase and encompasses steps 26 through 29.

In order for intrusion detection system 19 to be used, it must be first trained. The training phase is initiated at step 21. This is done by system administrator 10 flipping a switch (which may be located, for example, on database server 2 or on training module 4); by means of a preselected event occurring (e.g., the first of each month or the addition of a new table within database 1); or by any other means known to one of ordinary skill in the art for starting a software or other computer system.

At step 22, training module 4 observes, in real time, commands 5 that users 3 send to database 1. As used herein, "real time" means "during a short time interval surrounding an event". Thus, observing a command 5 in real time means that the command 5 is observed during a short time interval surrounding the instant that the command 5 enters the database 1.

Examples of commands 5 include querying the database 1, adding an entry to the database 1, deleting an entry from the database 1, and modifying an entry in the database 1. There are two major ways in which the observing step 22 is performed: real-time auditing and in-line interception.

Real-time auditing is typically used in cases where database 1 has an auditing feature. The auditing information may be placed into an audit table 11 internal to database 1 or into an external database log file 12. In real-time auditing, training module 4 instructs the database 1 to generate a new event every time a command enters database 1. Each event can include such items as the text of the command 5, a date/time stamp, information on the user that issues the command 5, the IP (Internet Protocol) address of the issuing computer 3, the application that issued the command 5, etc.

The event data can appear in string or binary form, and can be extracted using a number of different techniques, depending on the implementation of the IDS 19, including APIs (Application Programming Interfaces) that access the computer code 1. One example is to use ODBC (Open DataBase Connectivity), a set of C language API's that allows one to examine or modify data within database 1. If the Java programming language is used, JDBC (Java DataBase Connectivity) can be used instead. Another way of extracting the needed information from database 1 is to use code

injection or patching to inject logic into one or more modules 1,15 within database server 2, to transfer control to training module 4. In another embodiment, called "direct database integration", the database 1 vendor, who has access to the commands 5 in conjunction with the normal operation of the database 5, makes the commands 5 available to intrusion detection system 19. In yet another embodiment, in cases where database 1 supports it, external database log file 12 may be examined without the need to resort to special software. Once an event has been processed by the IDS 19, it can optionally be expunged from any table or log file it is stored in, to make room for subsequent events.

In in-line interception, at least one of a proxy, firewall, or sniffer 13 is interposed between database 1 and users 3. The proxy, firewall, or sniffer 13 examines packets of information emanating from users 3 and extracts the relevant information therefrom. Proxy, firewall, or sniffer 13 may need to decrypt the communications emanating from users 3 if these communications are encrypted.

After a command 5 has been captured in step 22, at step 23 training module 4 analyzes the command 5 and updates a set 6 of acceptable commands, again in real time. In other words, set 6 is updated commend-by-command. In one embodiment, the deriving step 23 comprises grouping the commands 5 into categories and

updating statistical information pertaining to the categories, all in real time. The categories are pre-selected, and can be fed to training module 4 via prestored but modifiable training parameters 17. The categories can include one or more of the following: canonicalized commands; the dates and times at which the commands access the computer code 1; logins (user IDs, passwords, catch phrases, etc.) of users 3 issuing the commands 5; the identities of users 3 issuing the commands 5; the departments of the enterprise in which the users 3 work, or other groups to which the users 3 belong; the applications (i.e., software programs or types of software programs) that issue the commands 5; the IP addresses of the issuing computers 3; frequency of issuing a given command 5 by a given user 3; identities of users 3 accessing a given field or fields within the computer code 1; the times of day that a given user or group of users 3 accesses a given field or fields within the computer code 1; the fields or combination of fields being accessed by given commands 5; tables or combinations of tables within the computer code 1 accessed by the commands.

The invention will now be further described with respect to the special case in which the commands 5 are grouped into categories comprising canonicalized commands. A canonicalized command is a command 5 stripped of its literal field data. Thus, for example, let us assume that the command 5 is:

SELECT NAME FROM PATIENTS WHERE NAME LIKE 'FRANK' AND AGE > 25

Literal field data is defined as a specific value of a parameter. In this case, the literal field data is "FRANK" and "25". Thus, a canonicalized form of the command 5 is:

SELECT NAME FROM PATIENTS WHERE NAME LIKE * AND AGE > *

Literal fields can include literal numbers (plain numbers), dates, times, strings, and potentially named ordinal values (symbolic words used to represent numbers, e.g., "January" represents the first month, "Finance" represents department 54, etc.). Training module 4 uses the canonicalized command as an index to group and categorize the commands 5. Even in a large database 1, the number of command categories is usually less than 20,000 or so.

To continue with our example, the following commands 5 would all be placed into the same command category:

SELECT NAME FROM PATIENTS WHERE NAME LIKE 'FRANK' AND AGE > 25

SELECT NAME FROM PATIENTS WHERE NAME LIKE 'BILL' AND AGE > 40

SELECT NAME FROM PATIENTS WHERE NAME LIKE 'ED' AND AGE > 25

In one embodiment, the command

SELECT NAME FROM PATIENTS WHERE NAME IS 'FRANK' AND AGE > 25

would be treated differently from the command

SELECT NAME FROM PATIENTS WHERE NAME LIKE 'FRANK' AND AGE > 25.

In an alternative embodiment, a list of synonyms is kept, so that "IS" is treated the same as "LIKE", and thus these two commands are treated as falling within the same command category.

In the last example, instead of using "ED", one could use "ED%" where % is a wildcard character. (Other wildcard characters might include #, ?, or _, depending upon the language variant used.) Thus, "ED%" would pick up "EDWARD", "EDDIE", and "EDWINA", as well as "ED". Wildcards are tracked by field and by command category, to give information needed to determine whether the use of the wildcard is innocuous or suspicious (see the following discussion with respect to step 25).

The fact that a wildcard is not observed in a given field during the training phase can be saved in the set 6 of acceptable commands, along with canonicalized data as described above. Then, during the operational phase, the presence of a wildcard character in that field is flagged as being suspicious.

For a given command 5, training module 4 examines the value of each literal field and determines whether its value is in the set of values seen thus far for that field, or whether it constitutes a new value. Continuing with our example based upon the three commands 5 given above, training module 4 stores the following information in NAME and AGE fields associated with the given canonicalized command category:

```
CANONICALIZED COMMAND CATEGORY:
   SELECT NAME FROM PATIENTS WHERE NAME LIKE * AND AGE > *
```

NAME statistics:
Total number of different field values: 3 (Frank, Bill, and Ed)
Generalize flag: FALSE (Do not generalize)
Wildcard flag: TRUE (wildcards have been used in this field, e.g. "ED")
AGE statistics:
Total number of different field values: 2 (25, 40)
Generalize flag: FALSE (Do not generalize)
Wildcard flag: FALSE (wildcards have not been used in this field)

Notice from the above that the actual names are stored because there aren't too many. Alternatively, all names having fewer than a preselected number of characters could be stored.

In one embodiment, whenever, for a given literal field, the number of observed values is greater than or equal to a preselected threshold value S, the training module 4 sets a flag for this field in this canonicalized command category, indicating that the field should be generalized, allowing any legitimate value. Thus, for example, if S=4 and training module 4 observes a fourth query:

SELECT NAME FROM PATIENTS WHERE NAME LIKE 'STEVE' AND AGE > 25,

training module 4 determines that it has already seen three distinct values (Frank, Bill, and Ed) in the NAME field, and it has just been presented with Steve. This now pushes the number of distinct cases for the NAME field to four, which meets or exceeds the preselected threshold. In this case, training module 4 automatically sets the ~~G~~generalize flag for this field in this

command category, indicating that any value is acceptable. The entry then changes to:

```
CANONICALIZED COMMAND CATEGORY:
   SELECT NAME FROM PATIENTS WHERE NAME LIKE * AND AGE > *
NAME statistics:
   Total number of different field values: 4 or more  (no
         actual field data saved)
   Generalize flag: TRUE (Allow any valid string for this
         field)
   Wildcard flag: TRUE  (wildcards have been used in this
         field, e.g. "ED")
AGE statistics:
      Total number of different field values: 2 (25, 40)
      Generalize flag: FALSE (Do not generalize)
      Wildcard flag: FALSE (wildcards have not been used in
         this field)
```

This technique may be advantageously used when there are a constrained number of values for that field. For example, if the field represents the month of the year, there are a maximum number of 31 days in any month, so S should be set to 31. Similarly, if the field represents the year, there are a maximum of 12 months in any year, so S should be set equal to 12. Once a given field within a given canonicalized command category has been generalized, there is no need to save all observed valued for this field anymore. Thus, the code intrusion detection system 19 uses less memory as more items are generalized.

In addition to the field attributes "generalize" and "wildcard", other attributes e.g., "length of string", can be used. The "generalize" attribute can be a partial or conditional "generalize", e.g., any name is allowed in the NAME field as long as it contains fewer than a preselected number of characters.

The training phase is ended, at step 24, by any one of a number of means. For example, system administrator 10 can flip a switch on database server 2 or training module 4. Alternatively, the training phase may end by a statistical technique, e.g., training module 4 monitors the occurrence or frequency of new canonicalized command categories being established or new fields being established, and determines that a preselected frequency threshold has been met. As with all of the preselected parameters, the preselected frequency threshold may be stored in training parameters 17. Alternatively, the training phase may end by the occurrence of a preselected elapsed or absolute time, or by any other means known to one of ordinary skill in the art.

At the conclusion of the training phase, the set 6 of acceptable commands that has been gradually built up in real time during the training phase becomes used as the basis for comparison during the subsequent operational phase (phase 3 in Figure 2). Based upon the above example, the set 6 of acceptable commands contains the following two commands:

SELECT NAME FROM PATIENTS WHERE NAME LIKE * AND AGE > 25

SELECT NAME FROM PATIENTS WHERE NAME LIKE * AND AGE > 40

In the above, * means that the field can hold any valid string value. These two commands in set 6 represent all valid SQL commands 5 observed during the training phase. This real-time system 4 has now built up a table of canonicalized commands,

determined which fields in these commands should be generalized, and has done this using a limited amount of memory. Even with 20,000 commands 5, only megabytes of storage are needed. This is in contrast to the off-line non-real-time systems of the prior art where logs 11, 12 may grow to gigabytes in length. For example, in the present invention, if the following command 5 was issued 5 million times:

SELECT NAME FROM PATIENTS WHERE NAME LIKE somename AND AGE > 25

it would require less than a kilobyte to represent this command data. On the other hand, off-line non-real-time training systems of the prior art would have to keep a full audit log 11, 12 of all 5 million occurrences as input to the off-line training process.

Returning to Figure 2, optional step 25 is invoked. When step 25 is used, suspicious activity is tracked during the training phase. For example, the number or percentage of times a wildcard character is used for a particular field for a particular command category can be tracked. If this number or percentage exceeds a preselected threshold (again, provided by parameters 17), the activity is deemed to be suspicious. This suspicious activity can then be reported to the system administrator 10. A wildcard may be either innocuous or suspicious, based upon where it is used. For example, if a wildcard is used in a NAME OF AUTHOR field in a database 1 of

library books, that is probably innocuous. On the other hand, if a wildcard is used in the PASSWORD field in a database 1 of credit card information, that most likely would be deemed suspicious. The system administrator can then selectively remove from the set 6 of acceptable commands such suspicious commands 5.

After the training phase and the optional suspicious activity reporting phase have been completed, the intrusion detection system 19 operates in the operational phase. At step 26, commands 5 that are currently accessing database 1 are compared by comparing module 7 against commands in the set 6 of acceptable commands. Module 7 can extract the current commands 5 in the same manner as described above for training module 4. The current command 5 is pre-processed similarly to the way the corresponding command 5 was pre-processed in the training phase, if this is needed to facilitate the comparison. For example, in the illustration described above, the command 5 is first canonicalized.

The basic rule by which module 7 operates is that if the current command 5 matches a command in the set 6 of acceptable commands, comparing module 7 allows the current command 5 to access database 1 (at step 27). If, on the other hand, the current command 5 does not match a command 5 in the set 6 of acceptable commands, comparing module 7 flags the current command 5 as being suspicious (at step 28). Then a post-flagging

protocol is performed at step 29. This protocol entails execution of at least one of the following steps: an alert is sent to the system administrator 10; the command 5 is not allowed to access the computer code 1; the command 5 is allowed to access the computer code 1, but the access is limited in some way (for example, the amount of data sent back to user 3 is limited, or wildcard characters are removed from certain fields); the command 5 is augmented, e.g., investigational code is inserted into the command 5 to provoke an audit trail; the user 3 sending the command 5 is investigated. Such an investigation can be performed by computer means (e.g., sending out a digital trace to determine the identity of the user 3) and/or by off-line means (sending a human private investigator to spy on user 3).

It will be seen from the above discussion that the present invention may offer, inter alia, the following advantages over prior art systems:

- It may reduce by several orders of magnitude the amount of data required to perform training of an intrusion detection system.
- It eliminates the need for the system administrator 10 to collect, manage, and delete large log files.
- It eliminates problems associated with overwritten log entries, ensuring that the training never misses any legitimate training data.

- It improves the performance of the database 1 by eliminating the storage of audit log data within built-in audit table 11.
- It eliminates the need for the system administrator 10 to manually configure database 1 to perform audit logging.
- It is an "on line" system, i.e., it doesn't disrupt the normal operations of the database 1.

The above description is included to illustrate the operation of the preferred embodiments and is not meant to limit the scope of the invention. The scope of the invention is to be limited only by the following claims. From the above discussion, many variations will be apparent to one skilled in the art that would yet be encompassed by the spirit and scope of the present invention.

What is claimed is: